計算機アーキテクチャ 119－13
（1996．8．27）

# UPCHMSのプログラムによる高速化手法

牧　晋広† 岡本秀輔† 曽和将容†

本研究室では，メモリの高速化の1手法として，キャッシュメモリに相当するメモリへのデータ転送を，プログラムにより1ワード単位で転送する階層メモリシステムについて提案をしてきた．本方式では，アプリケーションプログラムで必要とされるデータだけを，キャッシュメモリ相当のメモリに転送するため，従来のキャッシュメモリシステム以上に効率的なデータ転送が行える．
本稿では，このメモリシステムでの効率的なプログラミング法について述べ，その効果をシミュレーションにより評価を行う．

# The speedup strategy of programming for UPCHMS

NOBUHIRO MAKI , SHUSUKE OKAMOTO and MASAHIRO SOWA

We have proposed a new hierarchical memory system, which consists of level memories. This system has two kinds of programs. One is target program which executes arithmetic and logical part. Another is *data transfer programs* which control data mapping and transfer between different levels of hierarchical memories by one word. Furthermore, data transfers are done in parallel with the target program. Therefore, it is possible for this system to prepare necessary data into upper level memory before the target program refers to them.
In this paper, we propose optical programming strategies for this memory system and evaluate the programs which apply the strategies.

## 1. はじめに

計算機システムで，メモリの速度とプロセッサ速度のギャップを埋めるために，キャッシュメモリシステムが一般的に用いられている．キャッシュメモリシステム (以後CMSとよぶ) にはキャッシュミスが発生し，計算機システムの処理能力を下げる[5]．

我々はこの問題を解決するために，3階層に階層化されたメモリ間の内容をプログラムにより操作する方式を提案してきている[6),9)]．この方式では，データの振舞いを知るプログラマが各メモリ階層のデータの移動および配置を行うため，CMSで発生したようなキャッシュミスが発生しない．すなわち，アプリケーションプログラムが必要とするデータを，プログラムコントロールで，前もって上位階層のメモリに転送することが可能となり，キャッシュミスを無くすことができる．

本方式は，ソフトウェアで階層メモリを操作するという意味で，ソフトウェアプリフェッチ方式[3),4),7),8),11)]と類似しているが，対象としているメモリがキャッシュメモリでなく，各階層が独立にアドレスを持つメモリであることが大きく異なる．このようにキャッシュレベルのメモリに独立なアドレスを持たせることで，ソフトウェアプリフェッチが持っていた，データ挿入によるキャッシュラインの乱れ，プリフェッチしたデータが他のデータ転送のため使われないうちに再度主メモリに置換されるなどの現象を回避することができる．

本稿では，このメモリシステムのデータ転送を高速化するためのプログラミング手法について述べる．また，このメモリシステムを以後ユーザプログラム制御階層メモリシステム (User Program Controlled Hierarchical Memory System:UPCHMS) とよぶ．

## 2. ハードウェア構成

図1は，UPCHMSのブロック図である．PUはプロセッサユニット，IMはプログラムメモリ，DMはデータメモリを示している．DMは，レジスタに相当する超高速メモリ (VHM)，キャッシュメモリに相当する高速メモリ (HM)，主メモリ (MM) の3階層よりなる．

PUはVHMのみを対象として一般の演算を行う処理ユニットであり，HUはVHMとHM間のデータ転送，MUはHMとMM間のデータ転送をそれぞれ行なう転送ユニットである．転送ユニットは，それぞれIMに格納されたプログラムによってデータ転送を行

† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems,University of Electro-Communications

う．ユニット内の tc は，トークンカウンタとよばれるユニット間の同期機構である[10]．

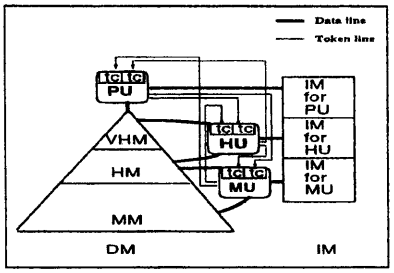



図1　UPCHMS のブロック図
Fig. 1　Diagram of UPCHMS

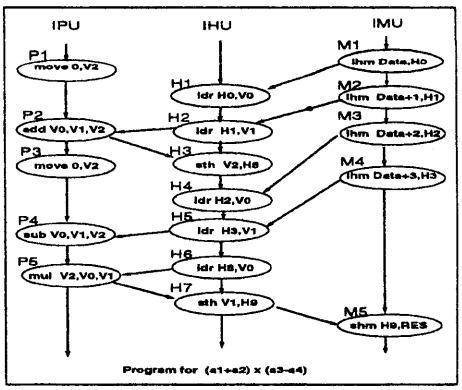



図2　UPCHMS のプログラム
Fig. 2　Program for UPCHMS

## 3. 動　　作

図2は MM の Data 番地から順番に格納されているデータ a1,a2,a3,a4 を用いて，(a1+a2)x(a3-a4) を計算するプログラムである．ここで IPU,IHU,IMU はそれぞれ PU,HU,MU 用の命令流であり，各命令間に付けられたアークは，実行の先行関係をあらわす．M2 と H2 のアーク上にある点は，トークンとよばれるもので，このトークンの送受により命令間で同期をとる．

今，図2で，P1,H1,M1 の命令の処理が終了しているものとする．この時，IPU の P2 命令で必要とされるデータ a2 は，MM の Data+1 番地から M2 命令によって HM に転送される．転送されたデータは，H2 命令によって VHM の V1 番地に転送され P2 命令により使われる．

IPU の命令が必要とするデータが，使用される直前に決定される場合，MM から VHM へのデータの転送が間に合わずに IPU 命令がデータを待つ状態になることがある．これはデータ待ち (Data Waiting:DW) とよばれており，この待ち時間は CMS のキャッシュミスペナルティに相当する．これは，階層メモリシステムが持つ性質であり，回避することは不可能である．この場合，UPCHMS では必要なデータを決定できた時点から転送を開始することができるので，この DW を最小にすることができる．これに対し，CMS ではキャッシュメモリにデータが無いことを検出してはじめてデータ転送を開始するので，待ち時間が相対的に長くなる．

## 4. 分　　岐

UPCHMS の分岐について無条件分岐と条件分岐を分けて述べる．

### 4.1 無条件分岐

UPCHMS の無条件分岐は，各ユニット毎に独立に行われる．すなわち，ユニット毎で無条件分岐命令を

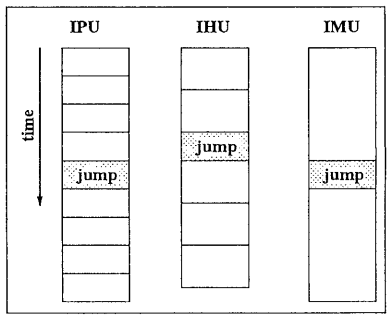



図3　無条件分岐

持つ．そのため，分岐処理は，ユニット毎で実行される時間に差が生じる．図3は，この無条件分岐の様子を示している．図で，四角は無条件分岐命令 jump とそれ意外の命令の実行を，縦方向は時間を示している．無条件分岐命令 jump は，IPU,IHU,IMU によって実行される時間が異なる．

### 4.2 条件分岐

条件分岐は，条件判定命令とブランチ命令に分けられる．条件判定命令は，IPU により行われ，分岐をするかどうかを決定し，その情報 (分岐コンディションフラグ) を IHU,IMU に転送する．ブランチ命令はすべてのユニットが持っており，IPU による分岐コンディションフラグが転送され初めて実行される．すなわち，分岐コンディションフラグが転送されるまでは，ブランチ命令はデコードされた時点でその処理が停止させられる．

図4は，条件分岐を示している．図3同様四角は命令の命令の実行を，縦方向は時間を示している．条件判定命令 less により IHU,IMU に分岐コンディションフラグが転送されている．IHU では，分岐コンディションフラグが IHU に転送される前に，ブランチ命令 branch が開始されたが，フラグが転送されていないために，処理が一旦停止し，フラグが転送された時

点で処理を再開している．

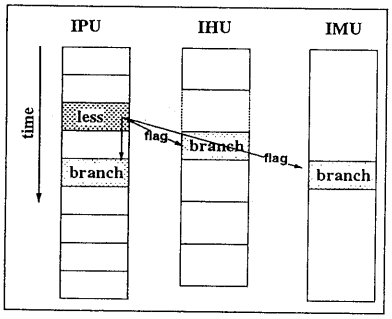



図4　条件分岐

## 5. プログラミングの最適化

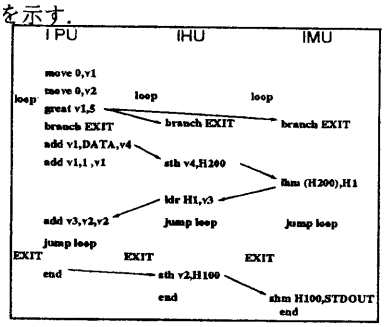
UPCHMS を有効に活用するためには，必要なデータが IPU の命令により使用される前に VHM に転送されているようにすればよい．これをするために，以下の方法が考えられる．

1 分岐判定の早期決定
2 次に必要となるデータを決定するための演算の早期化
3 データ参照の局所性を最大限利用する

上記の方法を UPCHMS のプログラムとそのトレース結果を見ることで説明する．

### 5.1 データの参照に局所性を示さないプログラム

MM の DATA 番地から順に格納されたデータを VHM に転送し，加算するプログラム add を例に話を進める．この add のプログラムを下図を示す．ここで，loop,EXIT は，ラベルを意味し，矢印は命令の依存関係を示す．



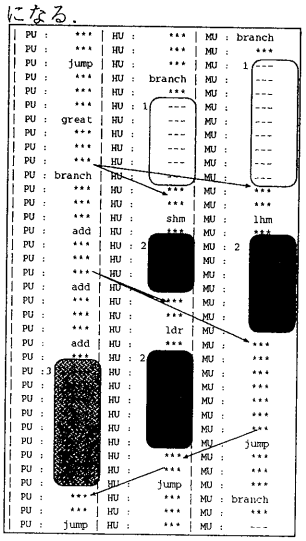

このプログラムは，IPU で次に必要となるデータの決定ができ次第，IHU( sth v5,H200 ),IMU( lhm (H200),H1 ) を用いて MU に知らせ，次のデータの転送を開始する．(IMU:lhm (H200),H1 と IHU:ldr H1,v3 の命令が相当する )

この add プログラムで逐次処理のプログラムでは以下のようになる．

逐次処理のプログラム

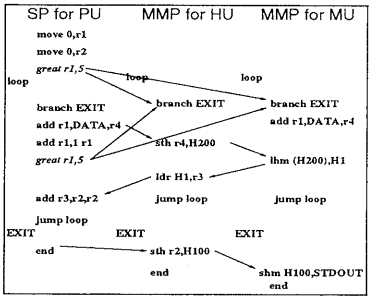
```
        move 0,r1
        move 0 r2
loop    great r1,5
        branch EXIT
        add r1,DATA,r4
        load r4,r3
        add r2,r3,r2
        add r1,1,r1
        jump loop
EXIT    store r2,STDOUT
        end
```

このUPCHMSプログラムの実行をトレースすると以下のようになる．



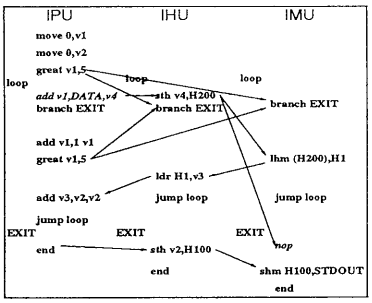
図は，左から順に PU，HU，MU の実行過程を，*** は命令がまだ実行途中であることを，四角で塗りつぶされた部分は，各ユニットの命令の待ちを示している．薄く塗りつぶされた四角 1( 左隅に 1 と記されたもの ) は，分岐判定が遅い為に，HU,MU が処理を停止していることを示している．これをここでは，分岐待ちと呼ぶ．また，濃く塗りつぶされた四角 2 は，データ決定が遅い為に，IHU,IMU の命令が，そのデータを転送処理を行うことができずに停止していることを示している．この待ちをここでは，転送待ちと呼ぶ．網目状の四角 3 は，分岐待ち，転送待ちにより，PU が必要なデータを得られずに処理を停止している状態であり，これがデータ待ちである．UPCHMS のプログラムを高速に処理させる為には，データ待ちの時間を少なくすれば良い．その為には，そのデータ待ちの原因となっている，分岐待ち，転送待ちの時間を減少させれば良い．

まずは，分岐待ちの時間を減少させる為のプログラム (分岐判定の早期化) を示す．



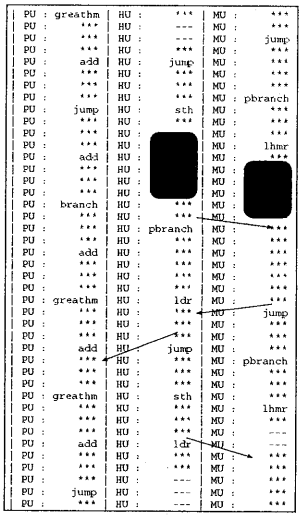

IPU に注目する．分岐判定命令 great が 2 個に増えていることがわかる．これは，以下の理由による．以前のプログラムでは分岐判定に必要な演算 add r1,1,r1 終了後，その結果をすぐには利用しないで，次のループ ( 2 命令後 ) の分岐条件判定命令 ( great r1,5 ) で利用していた．このことは，2 命令分分岐の判定が遅れることを意味し，無駄な分岐待ちを生じさせていた．そこで，分岐判定が実行される時期を早める方法として，命令 add r1,1,r1 の後すぐに分岐条件判定命令を挿入する (図中 *great v1,5*)．しかし，この操作だけでは，最初のブランチ命令で処理が停止してしまう．なぜならば，最初の実行では，分岐条件判定より前に，ブランチ命令を実行する為である．そこで，分岐命令を loop に入る前に，分岐判定命令を挿入する．これらの処理により，分岐条件成立が早まり，分岐待ちは，縮小される．

次に転送待ち時を減少させるためのプログラム (次に必要となるデータを決定するための演算の早期化) を示す．




これは，IPU が行っていたデータを特定する為の演算命令 ( *add v1,DATA,v4* ) をループの最上部に置くことで，PU が，HU,MU にデータを転送する時期が早くなる．これにより IMU の MM からのデータ転送時期が早められる．

そこで，このプログラムをトレースすると，次のような結果となった．




このプログラムでは，PU にデータ待ちがない．理由は，上記の 2 つの待ち時間を減少させた為である．すなわち分岐待ちの減少 ( 結果では，0 になっている ) と，転送待ちの減少である．HU,MU の待ちは，PU の演算処理よりも，データの転送処理の方が処理量が少ない為に，HU,MU が，PU の処理を待つ時間である．この待ちは，UPCHMS のプログラムの全体の実行時間には，影響しない．

この改善後のプログラムの実行結果ではデータ待ちがほとんど現れなかった．データ待ちが現れたのは演算結果の出力処理部であり，本質的な部分 (演算処理中) は，データ待ちは発生していない．

### 5.2 データの参照に局所性を示すプログラム

ここでは，行列積 (AB) のプログラムを例に述べる．今行列の 1 行 1 列分のデータ量が HM の容量を超えると仮定する．この場合の演算方針として，A の行要素を HM に配置するとすると，B の列要素は，1 データで十分である．なぜならば，HM に A の行要素を多く再利用しようと考えると，B の列要素は，一度使用するとしばらく A の次の行要素が HM に転送されるまで必要でなくなるためである．その B の列要素で転送されたデータが使用されるとすぐに次の B の列要素のデータを HM に上書き転送する．このようにデータ転送処理をすることで，行列積のプログラムが持つ局所性を有効に利用することができる．なぜならば，比較的短い時間内で繰り返し参照されるデータを HM に配置し，そうでないもので HM に転送されたデータはすぐに新しい他のデータと入れ換えるため，HM は局所性を示すデータと近い将来利用されるデータだけになるためである．このことは PU の演算処理に比べ，HU,MU のデータ転送処理を少なくするため，PU が使用するまでに HU,MU は，VHM にそのデータを転送することができることが多くなり，処理時間が短縮される．

## 6. 性能評価

UPCHMSの性能とプログラムの最適化による効果をみるため，シミュレーションによる評価を行う．

### 6.1 評価前提

シミュレータで設定したUPCHMS，CMSの各階層のメモリの容量およびアクセス転送時間を表1に，命令の実行時間を表2に示す．UPCHMSのHU，MUの命令実行時間は，メモリアクセス時間の影響を受けるため，表のように異なっている．CMSでは，ロードとストア以外の命令実行時間(CPU)を60とし，ロードとストア命令(load & store)は，キャッシュヒット時(hit)80とキャッシュミス時(miss)260という時間設定で行う．メモリ容量，アクセス時間の単位はそれぞれワード(1ワード32bit)，nsである．

命令メモリは，理想化を行いHMもしくはキャッシュメモリのアクセス時間で，1命令をフェッチできるものとする．

CMSは，UPCHMSのHMと同容量のキャッシュメモリを持ち，LRU置換，write through，フルアソシエィティブキャッシュ，1ブロック4ワードとしている[1),2)]．

表1 UPCHMS，CMSのメモリの容量および各階層のアクセス時間
Table 1 Size and access speed each hierarchical memory for UPCHMS and CMS.

| | | VHM,reg | HM,cache | MM |
|---|---|---|---|---|
| UPCHMS CMS | memory size | 16 | 128 | 65536 |
| | access time | 5 | 20 | 100 |

表2 UPCHMS，CMSの各種命令の実行時間(ns)
Table 2 Instructions' execution time for UPCHMS and CMS.

| | | | |
|---|---|---|---|
| UPCHMS | PU | HU | MU |
| | 60 | 60~65 | 155~175 |
| CMS | CPU | load&store | |
| | 60 | hit | miss |
| | | 80 | 260 |

評価プログラムとその特徴ををを表3に示す．評価プログラムは基本的に2つのプログラムを最適化したものとしてないものの4種類である．

### 6.2 シミュレーション結果およびその考察

#### 6.2.1 評価プログラム別の評価

図5は，UPCHMSとCMSの評価プログラム別の実行時間を示している．縦軸は，時間を表し、横軸は

表3 評価プログラムの名前，内容，特徴
Table 3 Benchmark programs.

| Program | Description | Characteristic |
|---|---|---|
| add1 | 数列加算1 | 最適化前の標準プログラム |
| add2 | 数列加算2 | 最適化後のプログラム |
| addC | 数列加算C | CMS用addプログラム |
| mul1 | 行列積算1 | 局所性を利用しないしたプログラム |
| mul2 | 行列積算2 | 局所性を利用したプログラム |
| mulC | 行列積算C | CMS用mulプログラム |

評価プログラムを示している．どのプログラムもCMSよりUPCHMSの実行時間の方が短縮していることがわかる．addプログラムは，前節で示した最適化手法1,2を用いて最適化を行っているが，最適化前と後では，22パーセント程度改善されていることがわかる．最適化後のプログラムでは，データ待ちが発生したのは，結果出力処理だけであった．

mulプログラムは，前節で示した最適化3の方法を用いて最適化を行っている．最適化前のプログラムも1,2の最適化は施してある．このプログラムでは，局所性を有効に利用することによる性能の改善をみることができる．結果では，最適化前と後で5パーセント前後改善された．

#### 6.2.2 評価の考察

addでは，最適化手法1,2をプログラムに適用することで，十分な性能向上を示した．これは，2つの最適化手法が，MM上のデータを必要とする命令の実行される時間間隔を十分に確保することを可能にするためである．この十分な時間間隔は，MMからVHMへのデータ転送の隠蔽を可能にする．その結果，結果出力処理以外はデータ待ち時間が0という結果にした．

mulでは，addの改善率に対して改善率が低い．これは，改善前の状態でも(最適化手法1,2は施してあるため)，HU,MUの並列データ転送がデータ待ちをある程度減らしていることが考えられる．このことは，改善前のプログラムとCMSの実行結果が，16パーセント程度改善されていることからもわかる．

## 7. おわりに

階層メモリ間のデータ転送をプログラムにより制御する，ユーザプログラム制御階層メモリシステム：UPCHMSのプログラムの最適化手法を述べた．UPCHMSは，HMの内容をプログラムで制御可能であるため，HMへのデータ配置および転送法が重要になる．また，アプリケーションプログラムと並列にデータ転送を行うことができるため，アプリケーションプログラムの演算処理とデータ転送処理が並行処理できる時間を多く確保することが重要である．

これらのことを考慮して，UPCHMSの性能を最大限引き出す手法として以下のものを提案した．

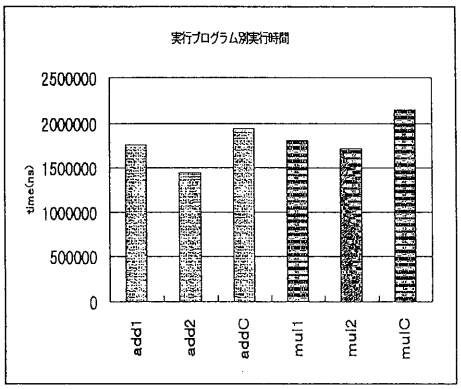



図5　評価プログラム別実行時間

1 分岐判定の早期決定
2 次に必要となるデータを決定するための演算の早期化
3 データ参照の局所性を最大限利用する

これらの手法による効果をみるために評価を行った．その結果では，最適化を施してないものとの比較で，最大22パーセント程度の改善を示した．今回の評価では，2種類のプログラムだけであるが，今後より多くのプログラムの改善および評価を行って良く予定である．

## 参考文献


1) Alvin, R. and A., D.: Cache Profiling and the SPEC Benchmarks:A CASE Study., *COMPUTER*, Vol. 27, No. 10, pp. 15–26 (1994).
2) Andrew, S.: *MINIX OPERATING SYSTEMS:Design and Implementation.*, Prentice Hall (1987).
3) Callahan, D., Kennedy, K. and Porterfield, A.: Software Prefetching, *Proc. the 4th International Conference on Architectural Support for Programming Languages and Operating Systems.*, pp. 40–52 (1991).
4) Callahan, D., Kennedy, K. and Proterfield, A.: An Architecture for Sortware-Controlled Data Prefetching., *Proc.the 4 th International Conference on Architectural Suport for Programming Langages and Operationg Systems*, pp. 40–52 (1991).
5) John, L. and David, A. P.: *COMPUTER ARCHITECTURE:A QUANTITATIVE APPROACH.*, Morgan Kaufmann (1990).
6) 牧晋広, 岡本秀輔, 曽和将容: ユーザプログラム制御階層メモリシステム, 修士論文, 電気通信大学情報システム学研究科 (1996).
7) Mowry, T. and Gupta, A.: Tolerating Latency through Software-Controlled Prefetching in Scalable Shared-Memory Multiprocessors., *Journal of Parallel and Distributed Computing*, Vol. 2, No. 12, pp. 87–106 (1991).
8) Mowry, T. and Gupta, A.and A, G.: Design and evalution of a compiler algorithm for prefetching., *Proc. of the 5th Intl.Conf. on Architectural Support for Programming Languages.*, pp. 62–73 (1992).
9) 佐藤正樹, 有田隆也, 曽和将容: 並列処理によるキャッシュ操作の明示化, 並列シンポジウム JSPP, Vol. 1, pp. 1–7 (1990).
10) 高木浩光, 河村忠明, 有田隆也, 曽和将容: 問題の持つ先行関係だけを保証する高速な静的実行順序制御機構, 並列情報処理シンポジウム JSPP, Vol. 1, pp. 57–64 (1990).
11) W.Y.Chen, R.A.Bringmann, S.A.Mahlke and J.E.Sicolo: An Effecient Architecture for loop Based Data Preloading., *Proc. of ACM MICRO-25*, pp. 92–101 (1992).